Lab #10

CSCI 4061 - Fall 2022 - 11/14/2022

Lab Preparation:

(1) Download lab files (git clone, git pull, or download from Canvas)

```
$ git clone https://github.umn.edu/csci-4061-fall-22/posted_labs.git
$ git pull
```

(2) Extract Lab Files from tar file

```
$ tar -xzvf lab_10_code.tar.gz
```

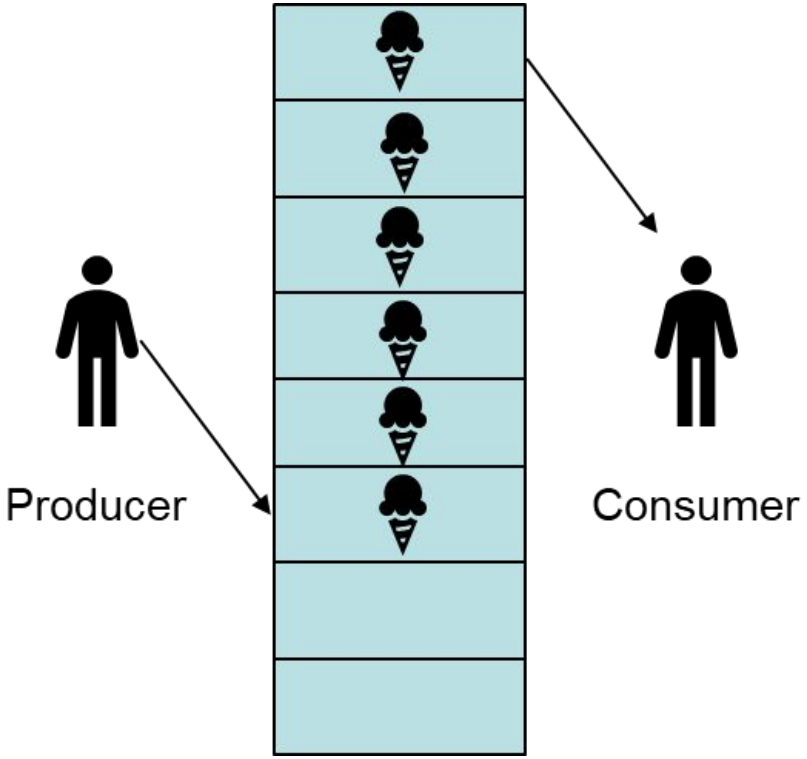
Lab Topics

- Syncrhonization
  - Condition Variables
- Exercise
- Project #3 Questions

Why do we need Condition Variables?
Producer/Consumer Problem (1)

- Producer puts items in the buffer
  - Locks buffer
  - Produces if space available, waits if full
  - Unlocks buffer

- Consumer pulls items from the buffer
  - Locks buffer
  - Consumes if item available, waits if empty
  - Unlocks buffer



Why do we need Condition Variables?
Producer/Consumer Problem (2)

- **busywait.c**
  - gcc -pthread -o busywait busywait.c
  - Run busywait
- **What is the problem of busywait.c?**
  - Consumes unnecessary CPU cycles
  - Depending on scheduling, if consumer thread never gets a chance to execute, count == BUFSIZE will always be true, producer thread may busy wait forever

Why do we need Condition Variables?
Producer/Consumer Problem (3)

- **How to fix the problem of busywait.c?**
- **Need richer synchronization**
  - **Producer**: if buffer full, "*I'm going to sleep, don't wake me up until condition buffer_not_full is met*"
    - After producing an item, signal buffer_not_empty condition
  - **Consumer**: if buffer empty, "*I'm going to sleep, don't wake me up until condition buffer_not_empty is met*"
    - After consuming an item, signal buffer_not_full condition
- **Read code:** condvar.c
  - Pay attention to the usage of pthread_cond_t, pthread_cond_wait, and pthread_cond_signal

pthread_cond_init

Name: **pthread_cond_init** - Initialize a condition variable

Protype:
```
int pthread_cond_init(pthread_cond_t *cond, pthread_condattr_t *attr);
```

Parameters:
`pthread_cond_t *cond` - Pointer to condition variable to init

`pthread_condattr_t *attr` - Pointer to condition var attributes

Returns: 0 → On Success

-1 → On Failure

Short-Hand:
```
pthread_cond_t cond = PTHREAD_COND_INITIALIZER;
```

pthread_cond_wait

Name: **pthread_cond_wait** - Block on a condition variable

Protype:

```
int pthread_cond_wait(pthread_cond_t *cond, pthread_mutex_t *mutex);
```

Parameters: `pthread_cond_t *cond` - Pointer to condition variable to wait for

`pthread_mutex_t *mutex` - Pointer mutex (lock) to acquire after condition is met

Returns: 0 → On Success

error # → On Failure

pthread_cond_signal

Name: **pthread_cond_signal** - Signal a condition to one thread

Protype:
```
int pthread_cond_signal(pthread_cond_t *cond);
```

Parameters: `pthread_cond_t *cond` - Pointer to the condition to signal

Returns: 0 → On Success

error # → On Failure

pthread_cond_broadcast

Name: **pthread_cond_broadcast** – Broadcast a condition to all threads

Protype:

```
int pthread_cond_broadcast(pthread_cond_t *cond);
```

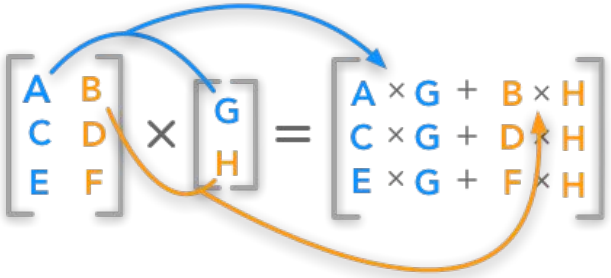
Parameters: `pthread_cond_t *cond` - Pointer to the condition to broadcast

Returns: 0 → On Success

error # → On Failure

Exercise: Parallel Matrix Multiplication

- Parallel matrix multiplication has a very wide application particularly for increasing speed of machine learning algorithms



Single Thread Matrix Multiplication:

$$\begin{matrix}1 & 2 & 3 \\ 4 & 5 & 6\end{matrix} * \begin{matrix}1 & 4 \\ 2 & 5 \\ 3 & 6\end{matrix} = \begin{matrix}[(1*1)+(2*2)+(3*3)] & [(1*4)+(2*5)+(3*6)] \\ [(4*1)+(5*2)+(6*3)] & [(4*4)+(5*5)+(6*6)]\end{matrix} = \begin{matrix}14 & 32 \\ 32 & 77\end{matrix}$$

Exercise: Parallel Matrix Multiplication

- **Multi-Threaded Matrix Multiplication**

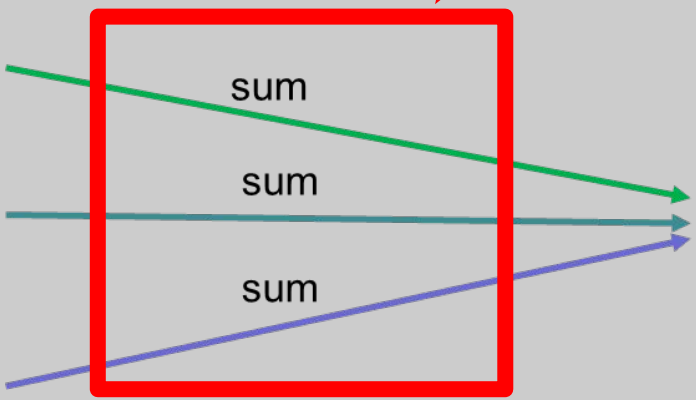
$$\begin{matrix}1 & 2 & 3\\ 4 & 5 & 6\end{matrix} * \begin{matrix}1 & 4\\ 2 & 5\\ 3 & 6\end{matrix} = \begin{matrix}[(1*1)+(2*2)+(3*3)] & [(1*4)+(2*5)+(3*6)]\\ [(4*1)+(5*2)+(6*3)] & [(4*4)+(5*5)+(6*6)]\end{matrix} = \begin{matrix}14 & 32\\ 32 & 77\end{matrix}$$

Bottleneck!

Thread 1: $\begin{matrix}[(1*1)] & [(1*4)]\\ [(4*1)] & [(4*4)]\end{matrix} = \begin{matrix}1 & 4\\ 4 & 16\end{matrix}$

Thread 2: $\begin{matrix}[(2*2)] & [(2*5)]\\ [(5*2)] & [(5*5)]\end{matrix} = \begin{matrix}4 & 10\\ 10 & 25\end{matrix}$

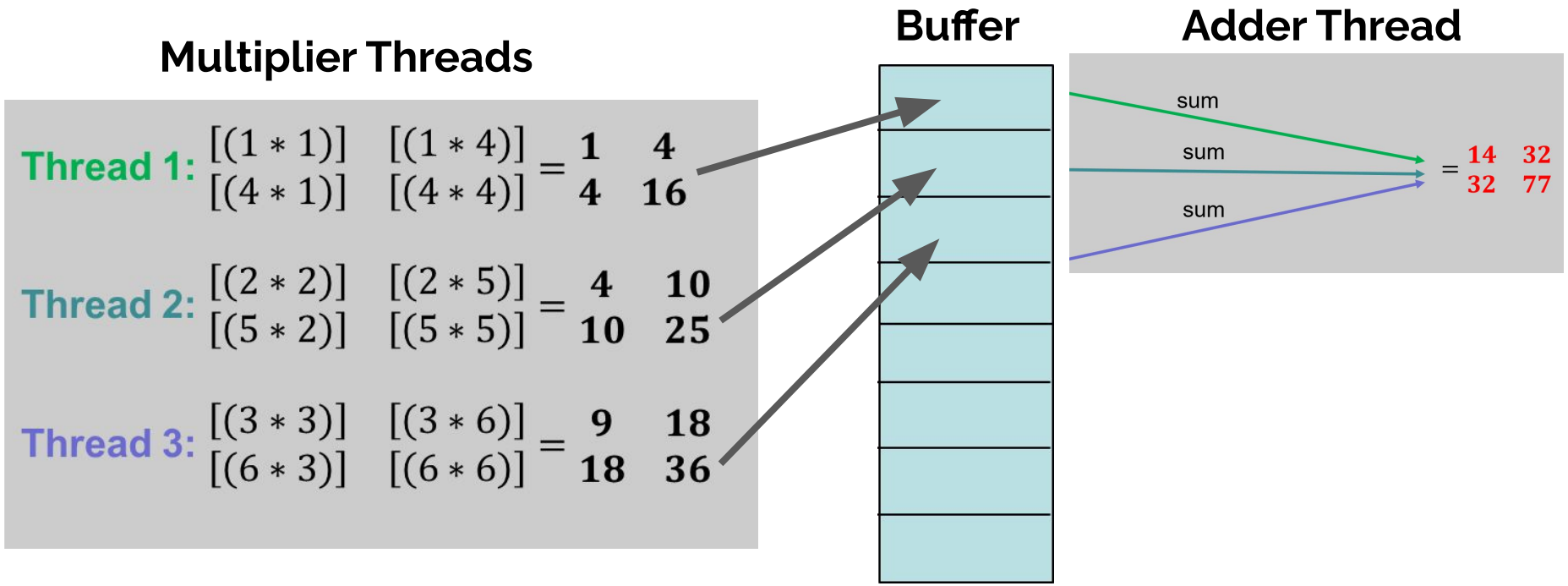
Thread 3: $\begin{matrix}[(3*3)] & [(3*6)]\\ [(6*3)] & [(6*6)]\end{matrix} = \begin{matrix}9 & 18\\ 18 & 36\end{matrix}$



$= \begin{matrix}14 & 32\\ 32 & 77\end{matrix}$

Exercise: Parallel Matrix Multiplication

- **Multi-Threaded Matrix Multiplication**



Exercise: Parallel Matrix Multiplication

- Using locks and condition variables, coordinate access to the “bounded_buffer” s.t. the answer is correct
  - Hint: Follow the TODO’s

How To Compile:

```
$ gcc -pthread -o matmult matmult.c
```

Good Output:

```
============== Correct Answer ==================
  4   7   4  10   6   2   3       7   2   7   4   5       252  206  218  162  170
  8   1   7   1   3   9   6       5   7   9   6   2       256  187  133  153  194
  8   4  10   5  10   2   4   x   9   9   3   5   1   =   336  216  216  177  177
  8  10   9   1   8   8   3       6   8   8   5   7       342  266  230  210  206
  1   1   6   6   9   1   5      10   2   4   2   4       233  143  134  113  131
                                  6   9   1   5  10
                                  7   1   3   4   6
================================================

SUCCESS: Matrices matched

============== YOUR  Answer ====================
252 206 218 162 170
256 187 133 153 194
336 216 216 177 177
342 266 230 210 206
233 143 134 113 131
================================================
```

Bad Output

```
…
ERROR! Matrix mismatch at index (2,3) Good Value [201] Bad Value [0]
ERROR! Matrix mismatch at index (2,4) Good Value [178] Bad Value [0]
ERROR! Matrix mismatch at index (3,0) Good Value [153] Bad Value [0]
ERROR! Matrix mismatch at index (3,1) Good Value [101] Bad Value [0]
ERROR! Matrix mismatch at index (3,2) Good Value [157] Bad Value [0]
ERROR! Matrix mismatch at index (3,3) Good Value [148] Bad Value [0]
ERROR! Matrix mismatch at index (3,4) Good Value [115] Bad Value [0]
ERROR! Matrix mismatch at index (4,0) Good Value [345] Bad Value [0]
ERROR! Matrix mismatch at index (4,1) Good Value [231] Bad Value [0]
ERROR! Matrix mismatch at index (4,2) Good Value [370] Bad Value [0]
ERROR! Matrix mismatch at index (4,3) Good Value [305] Bad Value [0]
ERROR! Matrix mismatch at index (4,4) Good Value [306] Bad Value [0]
ERROR: Matrices did not match, race condition detected

============== WRONG Answer ====================
  3  15   3  18  12
  5  25   5  30  20
  8  40   8  48  32
  4  20   4  24  16
 65  49  17  54  62
================================================
```

Bonus:

- If you time the single threaded and multi-threaded matrix multiplication functions you will notice that the parallel implementation is SIGNFICANTLY slower
  - Why would the parallel matrix multiplication be slower?
  - What ideas do you have for speeding up the implementation of parallel matrix multiplication?
- Add a comment to your Canvas submission answer these questions for 0.25 extra points on this lab